National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

Using Apache Science Data Analytics Platform from Jupyter

Presented By: Frank Greguska (JPL)

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, Pasadena, CA 91109-8099, U.S.A.

Overview

- Introduction to OceanWorks and Apache SDAP
- Jupyter Integration
- Example Jupyter Notebooks
- Future Development

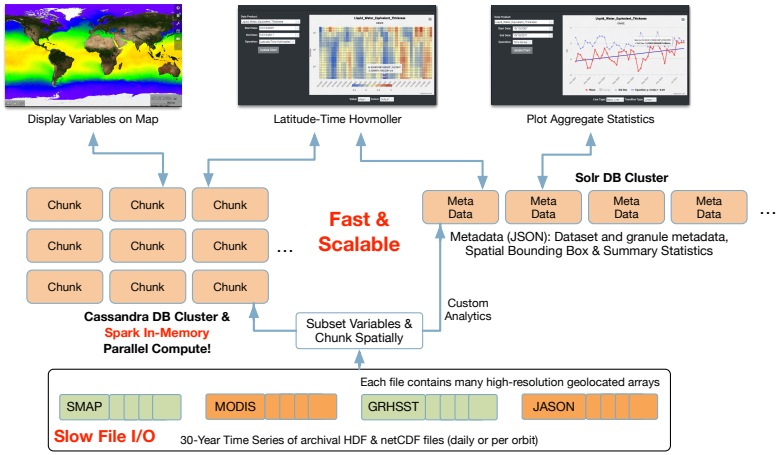
Apache Science Data Analytics Platform (SDAP)

- **OceanWorks** is to establish an **Integrated Data Analytics Center** at the NASA Physical Oceanography Distributed Active Archive Center (PO.DAAC) for Big Ocean Science
- Focuses on technology integration, advancement and maturity
- Collaboration between JPL, Center for Atmospheric Prediction Studies (COAPS) at Florida State University (FSU), National Center for Atmospheric Research (NCAR), and George Mason University (GMU)
- Bringing together PO.DAAC-related big data technologies
  - Big data analytic platform
  - Anomaly detection and ocean science
  - Distributed in situ to satellite matchup
  - Dynamic datasets ranking and recommendations
  - Sub-second data search solution and metadata translation and services aggregation
  - Quality-screened data subsetting
- All code open-sourced as Apache Science Data Analytics Platform (SDAP)

APACHE INCUBATOR

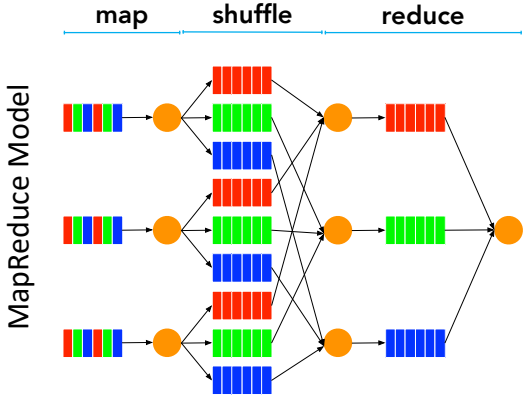
SDAP Cloud Analytics: NEXUS

- **NEXUS** is a data-intensive analysis solution using a new approach for handling science data to enable large-scale data analysis
  - Streaming architecture for horizontal scale data ingestion
  - Scales horizontally to handle massive amount of data in parallel
  - Provides high-performance geospatial and indexed search solution
  - Provides tiled data storage architecture to eliminate file I/O overhead
  - A growing collection of science analysis webservices



NEXUS' Two-Database Architecture

- **MapReduce**: A programming model for expressing distributed computations on massive amount of data and an execution framework for large-scale data processing on clusters of commodity servers. - J. Lin and C. Dyer, "*Data-Intensive Text Processing with MapReduce*"
  - **Map**: splits processing across cluster of machines in parallel, each is responsible for a record of data
  - **Reduce**: combines the results from Map processes



Jupyter Integration

- Python 3 module for easy integration
  - Source code: https://github.com/apache/incubator-sdap-nexus/tree/master/client
  - API Documentation: https://htmlpreview.github.io/?https://github.com/apache/incubator-sdap-nexus/blob/master/client/docs/nexuscli/nexuscli.m.html
- Exposes HTTP endpoints as functions
- Marshalls function input to JSON
- Unmarshalls server response to objects

```
def time_series(datasets, bounding_box, start_datetime, end_datetime,
                spark=False)
```

Send a request to NEXUS to calculate a time series.

datasets Sequence (max length 2) of the name of the dataset(s)
bounding_box Bounding box for area of interest as a `shapely.geometry.polygon.Polygon`
start_datetime Start time as a `datetime.datetime`
end_datetime End time as a `datetime.datetime`
spark Optionally use spark. Default: `False`

return List of `TimeSeries` namedtuples

SHOW SOURCE

```
def daily_difference_average(dataset, bounding_box, start_datetime,
                             end_datetime)
```

Generate an anomaly Time series for a given dataset, bounding box, and timeframe.

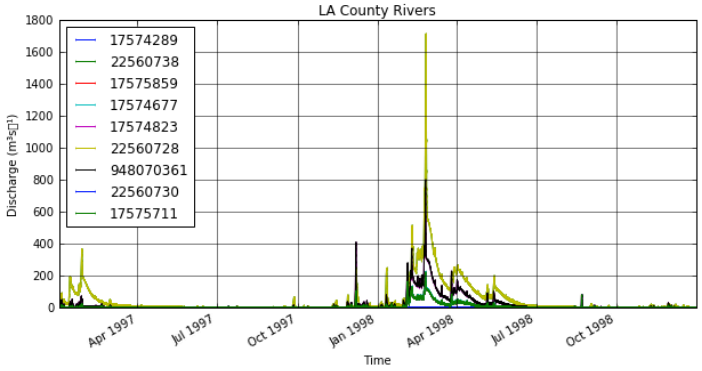
dataset Name of the dataset as a String
bounding_box Bounding box for area of interest as a `shapely.geometry.polygon.Polygon`
start_datetime Start time as a `datetime.datetime`
end_datetime End time as a `datetime.datetime`

return List of `TimeSeries` namedtuples

Subset

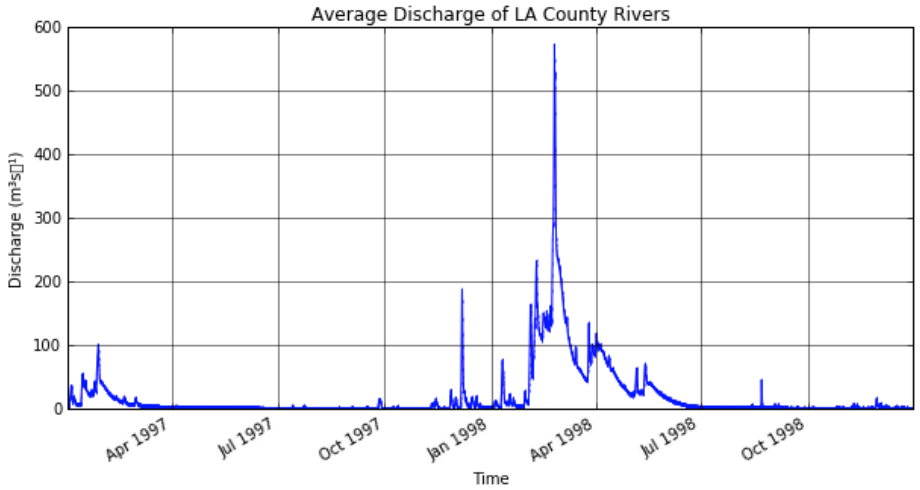
In [7]:
```
import requests
import json
import time
import nexuscli
from datetime import datetime
from pytz import UTC


nexuscli.set_target("https://oceanworks.jpl.nasa.gov", use_session=False)

ds = "RAPID_WSWM"
start_time = datetime(1997, 1, 1)
end_time = datetime(1998, 12, 31, 23, 59, 59)


# 10 Rivers in LA County
la_county_river_ids = [17575859, 17574289, 17575711, 17574677, 17574823,
                       948070361, 22560728, 22560730, 22560738]

la_county_river_data = dict()

start = time.perf_counter()
for rivid in la_county_river_ids:
    metadataFilter = "rivid_i:{}".format(rivid)
    result = nexuscli.subset(ds, None, start_time, end_time, None, metadataFilter)
    la_county_river_data[rivid] = result

print("Subsetting took {} seconds".format(time.perf_counter() - start))

show_plot([[point.time for point in points] for river, points in la_county_river_data.items()], # x values
          [[point.variable['variable'] for point in points] for river, points in la_county_river_data.items()], # y va
lues
          'Time', # x axis label
          'Discharge (m³s-¹)', # y axis label
          legend=[str(r) for r in la_county_river_data.keys()],
          title='LA County Rivers'
         )
```

```
Subsetting took 5.102821758016944 seconds
```



Custom Processing

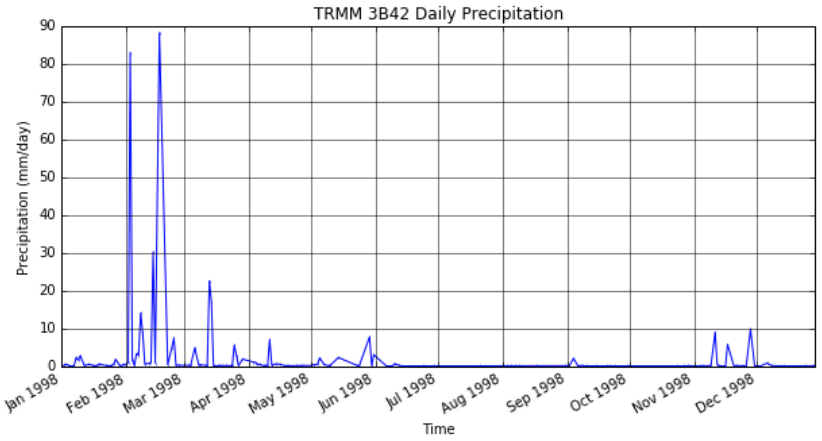
In [15]:
```
import numpy
%matplotlib inline
import matplotlib.pyplot as plt



discharge_rates = numpy.array([[tup[1] for tup in data[1]] for data in la_county_river_data])
single_river_time_steps = numpy.array([tup[0] for tup in la_county_river_data[0][1]])

avg_discharge_rates = numpy.mean(discharge_rates, axis=0)
show_plot([single_river_time_steps], # x values
          [avg_discharge_rates], # y values
          'Time', # x axis label
          'Discharge (m³s-¹)', # y axis label
          title='Average Discharge of LA County Rivers'
         )
```



Time Series

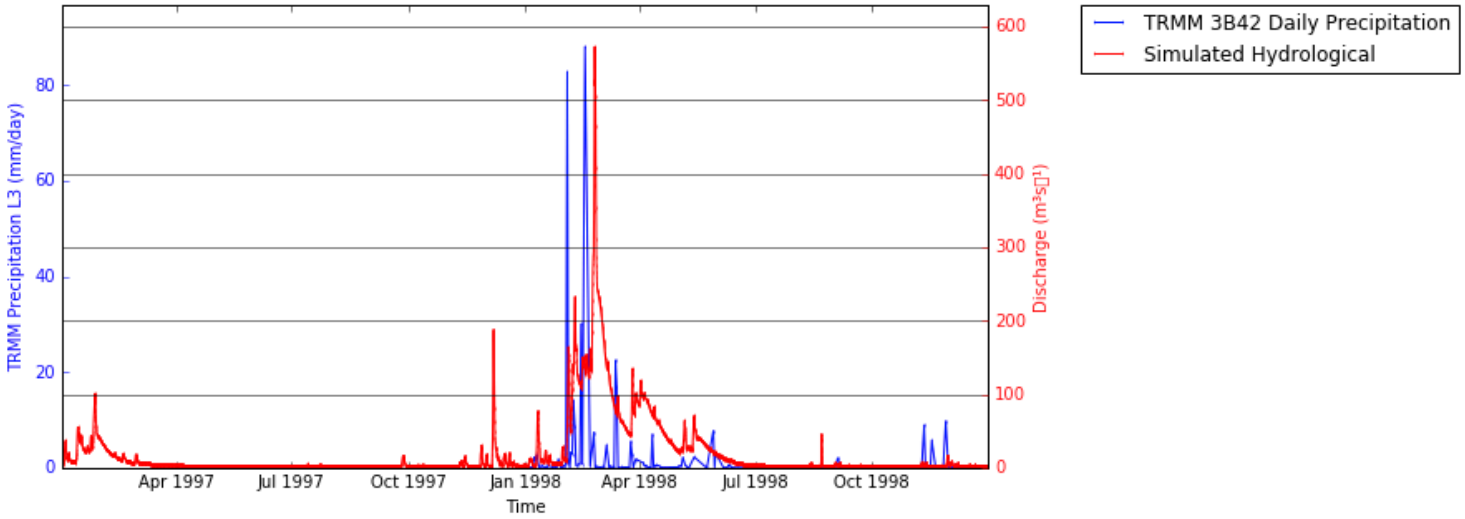
In [16]:
```
import time
import nexuscli
import shapely.wkt
from datetime import datetime

from shapely.geometry import box

nexuscli.set_target("https://oceanworks.jpl.nasa.gov")

la_county_wkt = \
    "POLYGON((-118.9517 34.8233, -117.6462 34.8233, -117.6462 32.7969, -118.9517 32.7969, -118.9517 34.8233))"

# TRMM Data only goes back to beginning of 1998
bbox = shapely.wkt.loads(la_county_wkt)
datasets = ["TRMM_3B42_daily"]
start_time = datetime(1997, 12, 31)
end_time = datetime(1998, 12, 31, 23, 59, 59)

start = time.perf_counter()
ts = nexuscli.time_series(datasets, bbox, start_time, end_time, spark=True)
trmm_ts = ts[0]

print("Time Series took {} seconds to generate".format(time.perf_counter() - start))

show_plot([trmm_ts.time], [trmm_ts.mean], 'Time', 'Precipitation (mm/day)', title='TRMM 3B42 Daily Precipitation')
```

```
Time Series took 0.4333303924649954 seconds to generate
```



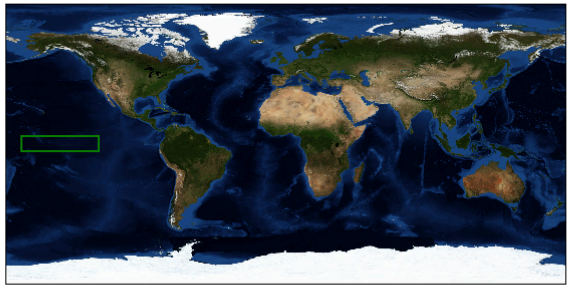
Comparing Time Series

In [17]:
```
show_plot_two_series(
    trmm_ts.time, single_river_time_steps,
    trmm_ts.mean, avg_discharge_rates,
    'Time', 'TRMM Precipitation L3 (mm/day)', 'Discharge (m³s⁻¹)',
    'TRMM 3B42 Daily Precipitation','Simulated Hydrological'
)
```

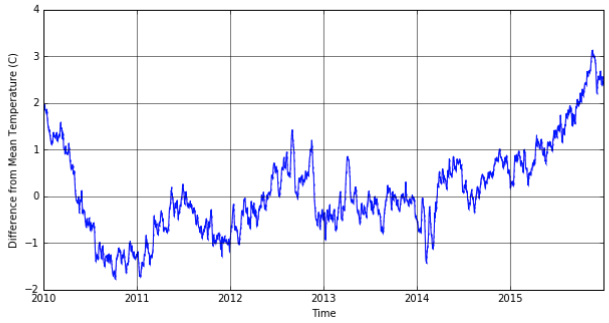


Difference from Mean

In [17]:
```
import time
import nexuscli
from datetime import datetime

from shapely.geometry import box

# Bounding box for the El Nino 3.4 Region
bbox = box(-170, -5, -120, 5)
plot_box(bbox)

start = time.perf_counter()

# Time range of interest
dataset = "AVHRR_OI_L4_GHRSST_NCEI"
start_time = datetime(2010, 1, 1)
end_time = datetime(2015, 12, 31)
# Call server
dda = nexuscli.daily_difference_average(dataset, bbox, start_time, end_time)

print("Daily Difference Average took {} seconds to generate".format(time.perf_counter() - start))

avhrr_dda = dda[0]
# Plot results!
show_plot(avhrr_dda.time, avhrr_dda.mean, 'Time', 'Difference from Mean Temperature (C)')
```

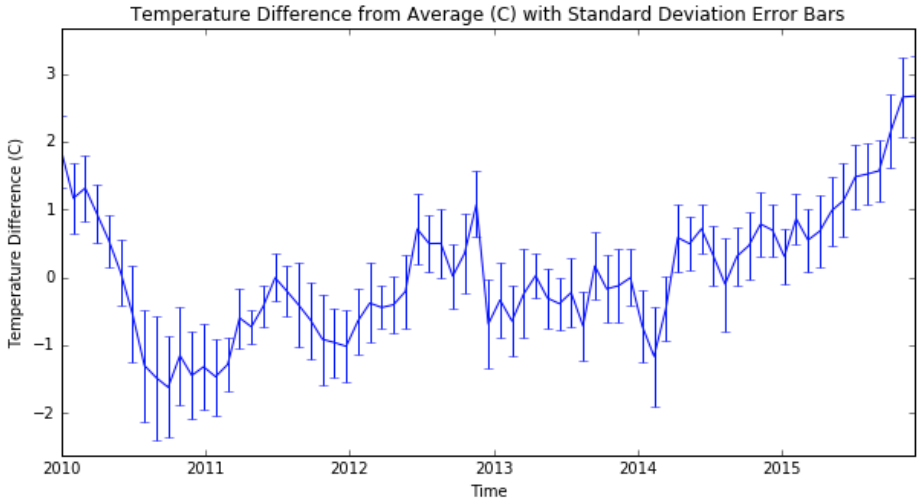


```
Daily Difference Average took 57.92217040248215 seconds to generate
```



Difference from Mean with Error Bars

In [32]:
```
# Sampling every 30th data point to reduce plot noise
means, dates, st_ds = avhrr_dda.mean[0::30], avhrr_dda.time[0::30], avhrr_dda.standard_deviation[0::30]

# Plot the extracted means
plt.figure(figsize=(10,5), dpi=100)
lines = plt.errorbar(dates, means, st_ds)
plt.xlim(dates[0], dates[-1])
plt.xlabel('Time')
plt.ylim(min(means)-1, max(means)+1)
plt.ylabel ('Temperature Difference (C)')
plt.title('Temperature Difference from Average (C) with Standard Deviation Error Bars', fontsize=12)
plt.show()
```

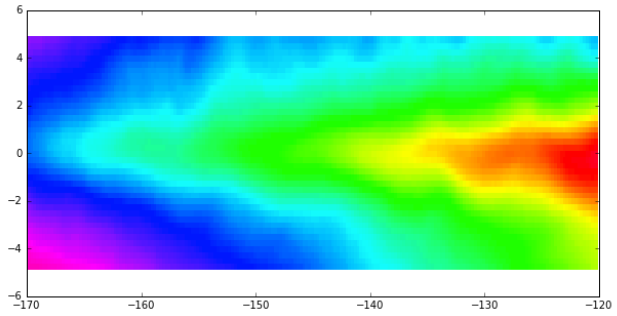


HTTP Access

In [54]:
```
import requests
import json
import datetime
import time
import numpy
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
from shapely.geometry import box
epoch = datetime.datetime.utcfromtimestamp(0)

# Build the HTTP request
host = "https://oceanworks.jpl.nasa.gov"

ds='AVHRR_OI_L4_GHRSST_NCEI'
bbox = box(-170, -5, -120, 5) #minx, miny, maxx, maxy
date_format = '%Y-%m-%dT%H:%M:%SZ'
startTime = int((datetime.datetime(2013,1,1) - epoch).total_seconds())
endTime = int((datetime.datetime(2013,10,30) - epoch).total_seconds())
request = "{}/timeAvgMapSpark?ds={}&startTime={}&endTime={}" \
          "&minLon={}&minLat={}&maxLon={}&maxLat={}&spark=local,16,32" \
    .format(host, ds, startTime, endTime, *bbox.bounds)
print (request)

# Send request to server
response = requests.get(request).json()

# Parse the response and create an image
lons = [point['lon'] for point in response['data'][0]]
lats = [a_list[0]['lat'] for a_list in response['data']]

my_list = numpy.ndarray((len(lats), len(lons)))
for x in range(0, len(lats)):
    for y in range(0, len(lons)):
        my_list[x][y] = response['data'][x][y]['avg']

norm = mpl.colors.Normalize(vmin=my_list.min(),vmax=my_list.max())

fig, ax1 = plt.subplots(figsize=(10,5), dpi=100)
ax1.pcolormesh(lons, lats, my_list, vmin=my_list.min(),vmax=my_list.max(), cmap='gist_rainbow')
```

```
https://oceanworks.jpl.nasa.gov/timeAvgMapSpark?ds=AVHRR_OI_L4_GHRSST_NCEI&startTime=1356998400&endTime=1383091200&mi
nLon=-170.0&minLat=-5.0&maxLon=-120.0&maxLat=5.0&spark=local,16,32
```

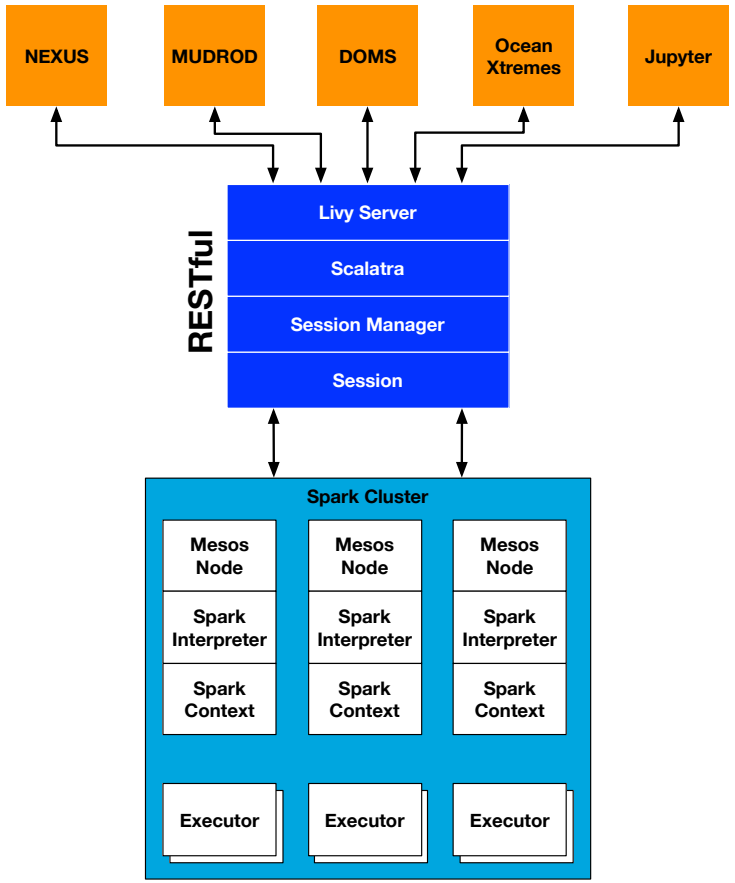
Out[54]:
```
<matplotlib.collections.QuadMesh at 0x7f152680eb70>
```



Current Development

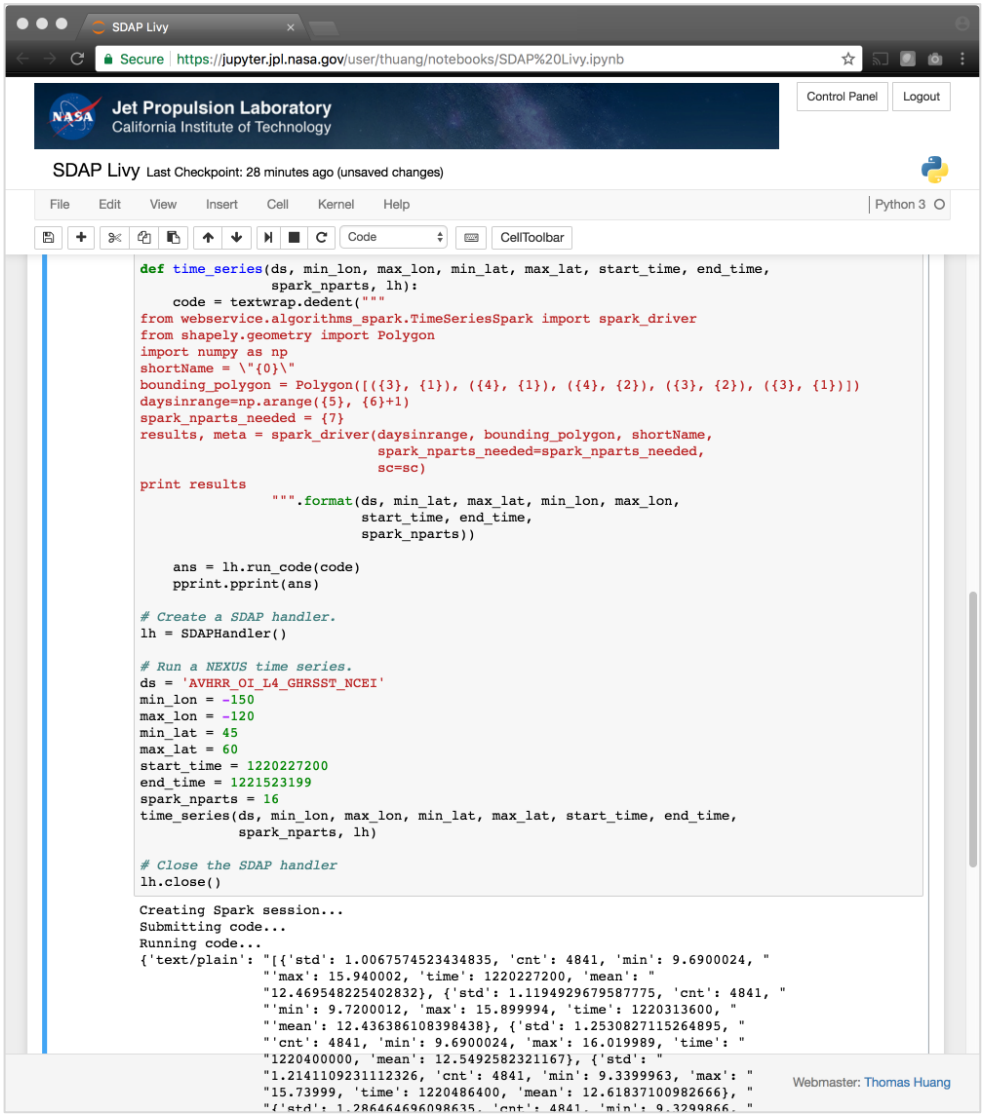
Current Development

- Developed independently, all the major services in OceanWorks require Apache Spark cluster
- If OceanWorks simply deploy these services to Amazon, it will require dedicated Apache Spark cluster for each
- Too many cluster and very costly, since Apache Spark recommends high memory machine instances
- Looking at the Amazon's EMR model. It is designed to be a job execution solution, and the jobs could from different applications
- Apache Livy provides a RESTful interface to Apache Spark cluster. It is a drop-in service to enable applications to interact with Spark cluster using RESTful api.
- The Apache Livy API also allows users to submit ad hoc map and reduce logics to be handled by the Spark cluster
- Through Apache Livy, scientists could use Jupyter environment to design their analytic algorithms that will be executed in the OceanWorks' Spark Cluster



Current Development

- Provide scientist a platform to develop algorithms to execute in OceanWorks' Spark cluster
- A new OceanWorks' RESTful service to offer flexible environment for researchers to experiment with their algorithms and our data, without having to deal with the complexity of Cloud and job management



Conclusion

- http://sdap.apache.org
- Questions?